PRÁCTICA 3: ESTRUCTURAS DE CONTROL CONDICIONALES

2 sesiones

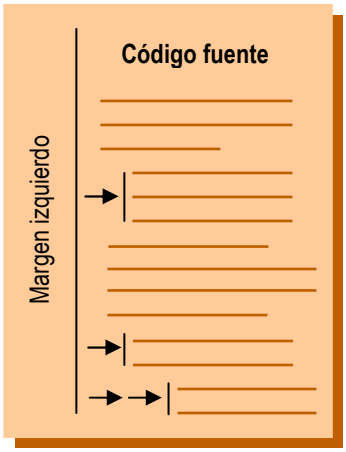
Semanas: del 23 y 30 de septiembre

OBJETIVOS:

- ✓ Revisar el concepto de algoritmo y entender la necesidad del diseño de algoritmos en el estudio y resolución de programas
- ✓ Comprender y diferenciar perfectamente los distintos tipos de instrucciones utilizados en el diseño de algoritmos
- ✓ Conocer y manejar con habilidad las sentencias de control condicionales

*No olvides que antes de pasar a la fase de diseño deberemos **analizar** exhaustivamente **el problema***

Recuerda también que la escritura de un programa exige la "**indentación**" o "**sangría**" del texto en el margen izquierdo de las diferentes líneas, lo que facilita el entendimiento y comprensión del programa realizado. Cualquier sentencia dentro del cuerpo de otra sentencia, debe sangrarse.



Ejercicio Resuelto 1. **Implementa un algoritmo en C que tenga como entrada el precio de venta de un piso, y que produzca como salida la comisión del vendedor, teniendo en cuenta que si el precio de venta es inferior a 100.000 €, la comisión es del 4%, si el precio está entre 100.000 € y 300.000 € la comisión es del 3% y si el precio es superior a 300.000 € la comisión es del 2'5 %.**

Solución a)

```
#include <iostream>
using namespace std;
int main()
{
    long int precio;
    float comision;

    cout << "\nIntroduzca el precio del piso:";
    cin >> precio;

    if  (precio < 100000)  {
         comision = precio * 0.04;
    }
    if  (precio >= 100000 && precio <= 300000)   {
         comision = precio * 0.03;
    }
    if  (precio > 300000)  {
         comision = precio * 0.025;
    }
    cout << "\nLa comision es de " << comision << "euros.";

}
```

En este algoritmo hay tres condiciones (la segunda de ellas compuesta) que se ejecutan secuencialmente. Independientemente de que este algoritmo produzca el resultado deseado, es mejorable en cuanto a su eficiencia, ya que tal y como está diseñado, las tres condiciones serán evaluadas siempre que ejecutemos el programa, ralentizando por tanto su ejecución.

Solución b)

```
#include <iostream>
using namespace std;
int main()
{
    long int precio;
    float comision;

    cout << "\nIntroduzca el precio del piso:";
    cin >> precio;

    if (precio < 100000)  {
        comision = precio * 0.04;
    }
    else  {
         if (precio < 300000)   {
              comision = precio * 0.03;
         }
         else  {
              comision = precio * 0.025;
         }
   }
    cout << "\nLa comision es de " << comision << "euros.";

}
```

En esta solución las tres condiciones son simples, y además, gracias a la estructura utilizada ("if-else" anidados), hemos conseguido una mayor eficiencia, ya que en el caso de que el precio sea inferior a 100000, sólo se evaluará una condición (la primera), ejecutándose a continuación la sentencia de salida. En cualquier caso, como máximo se evaluarán dos condiciones, ya que tanto si el precio es inferior o superior a 300000, sólo se evaluarán dos condiciones.

Solución c) Esta solución es idéntica a la anterior, y simplemente se propone para mostrar que los delimitadores de bloques solamente son necesarios cuando dentro del cuerpo del bloque "if" o del bloque

"else" hay más de una instrucción o de una estructura (tened en cuenta que a estos efectos, una estructura completa "if-else" es considerada como una sola instrucción). El algoritmo por tanto podría quedar así:

```
#include <iostream>
using namespace std;
int main()
{
    long int precio;
    float comision;

    cout << "\nIntroduzca el precio del piso:";
    cin >> precio;

    if (precio < 100000)
         comision = precio * 0.04;
    else
         if (precio < 300000)
             comision = precio * 0.03;
         else
             comision = precio * 0.025;
    cout << "\nLa comision es de " << comision << "euros.";

}
```

Solución d) Supongamos ahora que nos dicen que la mayoría de los precios de las viviendas son superiores a 300000 euros. Evidentemente esto no afecta a los cálculos que debe realizar el programa y por tanto seguirían siendo válidas las soluciones anteriores. Pero, ¿podríamos hacer algo por mejorar la eficiencia? Sí, y lo conseguiríamos cambiando el orden de las condiciones.

```
 #include <iostream>
using namespace std;
int main()
{
    long int precio;
    float comision;

    cout << "\nIntroduzca el precio del piso:";
    cin >> precio;

    if (precio >= 300000)
         comision = precio * 0.025;
    else
         if (precio >= 100000)
             comision = precio * 0.03;
         else
             comision = precio * 0.04;
    cout << "\nLa comision es de " << comision << "euros.";

}
```

De esta forma, en la mayoría de las ocasiones (cuando el precio sea superior a 300000) sólo se evaluará la primera condición. Observad cómo se han implementado las condiciones contrarias a las de la solución anterior.

Ejercicio Resuelto 2. **Implementa un algoritmo en C que realice las cuatro operaciones básicas de una calculadora (suma, resta, multiplicación y división). El programa debe leer los dos números y la operación y devolver el resultado.**

Solución a)

```
#include <iostream>

using namespace std;
int main()
{
     float a, b, resultado;
     char op;

     cout<<"\n Introduce el primer término: ";
     cin>> a;

     cout<<"\n Introduce el segundo término: ";
```

```
     cin>> b;

     cout<<"\n Introduce la operación";
     cin>>op;

     if (op == '+')
          resultado = a + b;
     else
          if (op == '-')
               resultado = a - b;
          else
               if (op == '*')
                    resultado = a * b;
               else
                    if (op == '/')
                         resultado = a / b;

 cout<<"El resultado es: "<<resultado<<"\n";

}
```

Solucion b)

```
#include <iostream>

using namespace std;
int main()
{
     float a, b, resultado;
     char operacion;

     cout<<"\n Introduce el primer término: ";
     cin>> a;

     cout<<"\n Introduce el segundo término: ";
     cin>> b;

     cout<<"\n Introduce la operación";
     cin>>op;

     switch (operacion){
          case '+': resultado = a+b;
                    break;
          case '-': resultado = a - b;
                    break;
          case '*': resultado = a * b;
                    break;
          case '/': resultado = a / b;
                    break;
     }

     cout<<"El resultado es: "<<resultado<<"\n";

}
```

En la solución b se ha utilizado la sentencia switch que permite escoger entre múltiples alternativas. Obtenemos así una solución más clara y eficiente que con la primera opción.

Ejercicio Resuelto 3. **Realiza un programa que calcule en índice de masa corporal de una persona. Se debe introducir el peso en kilogramos y la altura en metros, el índice de masa corporal se calcula con la siguiente fórmula: peso/(altura*altura). El programa debe devolvernos el tipo de peso que tenemos con respecto al IMC teniendo en cuenta la siguiente tabla.**

IMC	Tipo de peso
< 18.0	Inferior al normal
18.1 – 24.9	Normal
25.0 – 29.9	Sobrepeso
> 30.0	Obesidad

Ejemplo 1:

```
Introduce tu peso en Kg:
```

```
75
Introduce tu talla en m:
1.80
Tu IMC es: 23.1481. Normal
```

Solucion

```
#include <iostream>
using namespace std;
int main()
{
    float imc, altura, peso;


    cout << "Introduce tu peso ";
    cin >> peso;
    cout << "Introduce tu altura en metros ";
    cin >> altura;
    imc = peso/(altura*altura);

    if (imc<=18.0)
       cout<<"Tu IMC es:"<<imc<<" Peso inferior al normal"<<endl;
    else if (imc<25)
         cout<<"Tu IMC es:"<<imc<<"Peso normal"<<endl;
     else if (imc< 30)
           cout<<"Tu IMC es:"<<imc<<"Sobrepeso"<<endl;
          else cout<<"Tu IMC es:"<<imc<<"Obesidad"<<endl;
}
```

Ejercicio Propuesto 1. **Diseña un programa que pida introducir dos números por teclado y nos indique cual es el mayor y cual es el menor.**

Ejercicio Propuesto 2. **Diseña un programa que pida un número por teclado y nos diga si es par o impar, es decir, si es divisible por dos o no.**

Ejercicio Propuesto 3. **Diseña un programa que gestiona el color de las cunas en una maternidad (Azul para niños y Rosa para niñas). El programa nos pide el sexo del bebé. (M / F) y dependiendo del sexo del bebé le asigne el color de la cuna.**

Ejercicio Propuesto 4. **Realiza un programa de dados 3 números enteros nos diga cual es el mayor, cual es el menor y cual es el de en medio.**

Ejercicio Propuesto 5. **Las entradas del parque de atracciones, varían con respecto a la altura de las personas, además nos van a entregar una pulsera con el color de las atracciones a las que podemos subir, siguiendo el criterio:**

Menos de 1.00 m → Atracciones para bebes. Pulsera de color Blanco.

Entre 1.00m y 1.20m → Atracciones infantiles. Pulsera de color Amarillo.

Entre 1.20m y 1.40 → Atracciones junior. Pulsera de color Naranja.

Más de 1.40 → Atracciones adulto. Pulsera de color Rojo.

Ejemplo:

```
Introduce tu altura: 1.35
Junior. Pulsera Naranja
```

Ejercicio Propuesto 6. **Implementa un algoritmo que lea un carácter e indique si es o no una letra. Además, en caso de ser letra, debe indicarse si es vocal o consonante y si está en minúsculas o mayúsculas. Ten en cuenta que en lenguaje C la comparación de caracteres utiliza el código ASCII asociado a cada carácter. Sugerencia: buscar en Internet o en un libro "tabla códigos ASCII".**

Ejercicio Propuesto 7. **Implementa un algoritmo que lea una fecha (por separado el día, el mes y el año) y muestre un mensaje indicando si la fecha es correcta o no (Se recuerda que son**

bisiestos los años múltiplos de 4, excepto los múltiplos de 100, y que sí son bisiestos los múltiplos de 400). En el caso de que sea correcta, deberá mostrar el día siguiente.

Ejercicio Propuesto 8. **En una empresa se tiene el registro de la hora de entrada y salida de cada empleado. Escribe un programa en C que pregunte al usuario la hora y minuto de entrada, así como la hora y minuto de salida y calcule y escriba las horas y minutos que se han trabajado.**

Ejemplo 1:

```
Introduce la hora de entrada:
7
Introduce los minutos de entrada:
12
Introduce la hora de salida:
14
Introduce los minutos de salida:
23
El tiempo trabajado es: 7 horas y 11 minutos
```

Ejemplo 2:

```
Introduce la hora de entrada:
22
Introduce los minutos de entrada:
50
Introduce la hora de salida:
5
Introduce los minutos de salida:
10
El tiempo trabajado es: 6 horas y 20 minutos
```

Ejercicio Propuesto 9. **Escribe un programa en C que dibuje un cuadrado o un rectángulo en función de la elección del usuario. Para ello, el programa preguntará al usuario si desea dibujar un cuadrado o un rectángulo. En función de la respuesta solicitará el tamaño de un lado (para el cuadrado) o de dos lados (para el rectángulo).**

C no tiene instrucciones estándar para dibujar en pantalla. Por tanto, vamos a usar una librería gráfica adicional.
Puedes mirar en el apéndice de la práctica los comandos para dibujar en pantalla que ofrece la librería. En lugar de esos, que usaremos en otros ejercicios, aquí emplearemos lo que se llaman "gráficos de tortuga". Consisten en que dibujamos moviendo un cursor gráfico situado en un punto de la pantalla (que en la versión original era una tortuga, ya que los gráficos de este tipo nacieron para enseñar a niños). Podemos hacer las siguientes operaciones:

- **Dibujar avanzando en línea recta en la dirección en que está mirando ahora la tortuga:** `draw(longitud)`. **Inicialmente, la tortuga está mirando hacia la derecha (0 grados).**
- **Cambiar la orientación de la tortuga haciendo que gire sobre sí misma un cierto número de grados:** `turn(grados)`**. Para girar a la izquierda hay que hacerlo una cantidad negativa, y a la derecha, positiva.**
- **Colocar la tortuga en la posición inicial** `(x,y)` **que queramos:** `setTurtlePos(x,y)`

Así por ejemplo podríamos dibujar los cuatro lados de un cuadrado repitiendo cuatro veces las siguientes órdenes

```
//dibujar en línea recta 100 pixeles
draw(100);
//girar 90 grados hacia la derecha (positivo es a derecha, negativo a
izquierda)
turn(90);
```

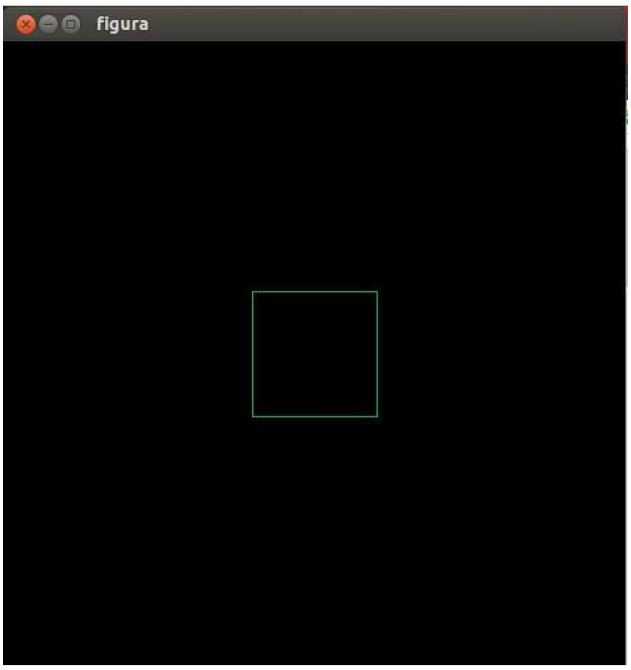
Cuidado, para compilar los ejemplos gráficos necesitas una orden distinta a la habitual, ya que estamos usando una librería adicional

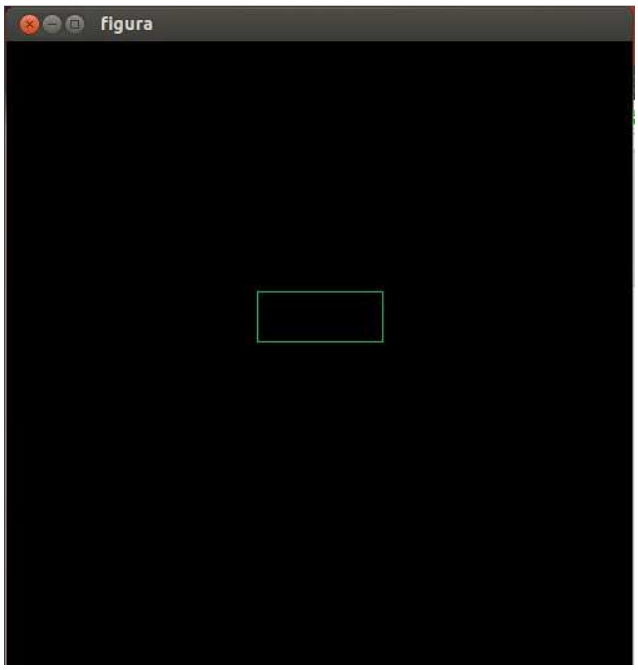
```
g++ -o figura figura.c gfx.c -lX11
```

Deberás descargarte también los ficheros `gfx.c` **y** `gfx.h` **desde Moodle.**

Además del código para resolver el ejercicio planteado deberás escribir las siguientes líneas de código:

```
#include "gfx.h"  // fichero con las funciones gráficas

...

int main() {

char c;
...
//instrucción para abrir una pantalla para dibujar
gfx_open(500, 500, "figura");
// instrucción para indicar el color del dibujo
gfx_color(0,200,100);
// situación del punto de inicio del dibujo
setTurtlePos(200,200);
...

...

//finalizar la ejecución y cerrar la pantalla de dibujo
c = gfx_wait();
}
```





ERRORES DE PROGRAMACIÓN COMUNES

No utilizar secuencias de sentencias delimitadas entre llaves. Prescindir de las llaves cuando se programan secuencias de control puede dar lugar a errores difíciles de detectar y depurar.

Incluir un ';' junto a la sentencia de control. No es correcto escribir un ';' después de la condición de un if. Ejemplo:

```
if (contador == 0);
    contador = var+2;
```

Independientemente del resultado de la evaluación de la condición del if, siempre se ejecuta la sentencia de asignación. ¿Por qué? Pues porque el ';' de detrás indica una sentencia nula en el cuerpo del if, es decir, no hacer nada en dicho cuerpo. La sentencia de asignación estaría fuera del cuerpo del if.

Confusiones entre asignación y comparación de igualdad Este error es muy frecuente y difícil de detectar. Si se escribe

```
if (contador = 0)
```

Se está asignando 0 al contador, no se está haciendo una comparación.
Recuerda que el operador de comparación de igualdad es ==

```
if (contador == 0)
```

Comprobar que una misma variable cumpla más de una restricción.

Si escribimos

```
if (contador>=3 && <6)
```

Vamos a tener un error de compilación puesto que la comparación <6 no sabe a qué variable se refiere. Aunque sea la misma que se acaba de escribir hay que volver a repetirla.

La sentencia correcta es:

```
if (contador>=3 && contador<6)
```

Apéndice: cómo dibujar en pantalla con gfx.c

Para dibujar en pantalla dispones de una librería muy simple llamada gfx.c que además de los gráficos de tortuga que ya hemos visto en los ejercicios nos permite hacer una serie de operaciones básicas:

- Abrir una ventana donde dibujar: `gfx_open(ancho_ventana, alto_ventana, titulo_ventana).` El sistema de coordenadas es como el de ejes cartesianos, la x es horizontal, la y vertical. El origen de coordenadas (x=0,y=0) está en la esquina superior izquierda de la ventana.
- Cambiar el color en que se va a dibujar como combinación de rojo, verde y azul: `gfx_color(valor_rojo, valor_verde, valor_azul).` Por ejemplo `gfx_color(255,0,0) es rojo.`
- Dibujar una línea sabiendo su (x,y) inicial y final: `gfx_line(x1,y1,x2,y2).` Para dibujar un punto simplemente puedes usar la misma x e y como coordenada inicial y final.

- Las instrucciones de dibujar en pantalla no tienen efecto hasta que no se hace `gfx_flush().` Es decir, si quieres que aparezca un objeto formado por varias líneas dibújalas todas y luego haz `gfx_flush()` para que aparezca.
- Borrar la pantalla: `gfx_clear().`

Instalar la librería gráfica

Este apartado solo se aplica en caso de que estés trabajando en tu propio ordenador. Los ordenadores del laboratorio ya tienen hecha la instalación necesaria

Es posible que no tengas instalado en tu Ubuntu lo necesario para compilar estos programas gráficos. Necesitas el paquete 'xorg-dev'. Teclea en una terminal

```
sudo apt-get install xorg-dev
```

Te pedirá la contraseña de administrador. En la máquina virtual que usamos para la asignatura es "p1" (sin las comillas).

Compilar programas

Para compilar los programas que hagan uso de la librería **en linux** necesitarás añadir un par de parámetros al final de la orden que usamos siempre:

```
g++ -o cuadrado cuadrado.c gfx.c -lX11
```